6.2 stacks, queues, induction proofs

Thursday, October 6, 2022

Today:
- wrap up stacks and queues
- induction proofs
  - on summation functions
  - on recursive programs

STACKS

push / pop
operate at one end

LIFO
implemented as LL
depth-first search

QUEUES

enqueue / dequeue
enqueue at back
dequeue at front

FIFO
implemented as LL
breadth-first search

INDUCTION PROOFS

Consider some algorithm with double-nested loops:

```
for i = 0 ... n
    for j = 0 ... n-i
        print i*j
```

We previously said the number of times the print statement is performed is

$$n + (n-1) + (n-2) + \cdots + 3 + 2 + 1$$

And we previously claimed

$$\sum_{i}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Let's FORMALLY prove this!

Proof by induction is a multistep process:

You must include all these steps:

① BASE CASE
Show statement is true for smallest value.

② INDUCTIVE HYPOTHESIS
Assume statement is true up to some constant k.

③ INDUCTIVE STEP
Show statement is true for the next value $k+1$ using the assumption from step 2.

Why does this process work? (as a rigorous & convincing argument)

Think of a long line of dominoes.

① Base case: the first domino falls

② Inductive hypothesis: assume a middle domino falls

③ Inductive step: show the next domino must also fall

If we've written an inductive proof, we don't have to wait around to see if domino #1,000,000 falls – we've been convinced already!

Note: Induction is only valid with integers and must have a lower bound.

example problem 1: Prove that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

① base case:

$n = 1$

LHS is $\sum_{i=1}^{1} i = 1$ RHS is $\frac{1(1+1)}{2} = 1$

$1 = 1$ ✓

② inductive hypothesis:
Assume that if $n=K$, formula is true: $\sum_{i=1}^{K} i = \frac{k(k+1)}{2}$

(2) inductive hypothesis:
Assume that if $n=k$, formula is true: $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$.

(3) inductive step:
Want to show that if $n=k+1$, formula is true: $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+1+1)}{2}$

Let's start with LHS:

$$\sum_{i=1}^{k+1} i = \left(\sum_{i=1}^{k} i\right) + k+1 \quad \text{by separating out the last term}$$

$$= \left(\frac{k(k+1)}{2}\right) + k+1 \quad \text{by using the inductive hypothesis}$$

$$= \frac{k(k+1) + 2(k+1)}{2} \quad \text{by math}$$

$$= \frac{(k+1)(k+2)}{2} \quad \text{also math}$$

example problem 2: Prove that $\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$

(1) base case: $n=1$

LHS: $\sum_{i=1}^{1} \frac{1}{i(i+1)} = \frac{1}{1\cdot(1+1)} = \frac{1}{2}$

RHS: $\frac{1}{1+1} = \frac{1}{2}$ equal ✓

(2) inductive hypothesis:
Assume that for some integer k, $\sum_{i=1}^{k} \frac{1}{i(i+1)} = \frac{k}{k+1}$.

(3) inductive step:
Want to show $\sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \frac{k+1}{k+1+1} = \frac{k+1}{k+2}$.

Starting with LHS:

$$\sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \left(\sum_{i=1}^{k} \frac{1}{i(i+1)}\right) + \frac{1}{(k+1)(k+1+1)}$$

(the last term of the sum, $i = k+1$; by separating the last term of the sum)

$$= \frac{k}{k+1} + \frac{1}{(k+1)(k+2)} \quad \text{by applying inductive hypothesis}$$

$$= \frac{k\cdot(k+2)}{(k+1)(k+2)} + \frac{1}{(k+1)(k+2)} \quad \text{by math}$$

$$= \frac{k^2+2k+1}{(k+1)(k+2)} = \frac{(k+1)(k+1)}{(k+1)(k+2)} \quad \text{by math}$$

$$= \frac{k+1}{k+2} \quad ✓$$

How can we use induction on algorithms?

Consider the mathematical function factorial:

$n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots (2) \cdot (1)$

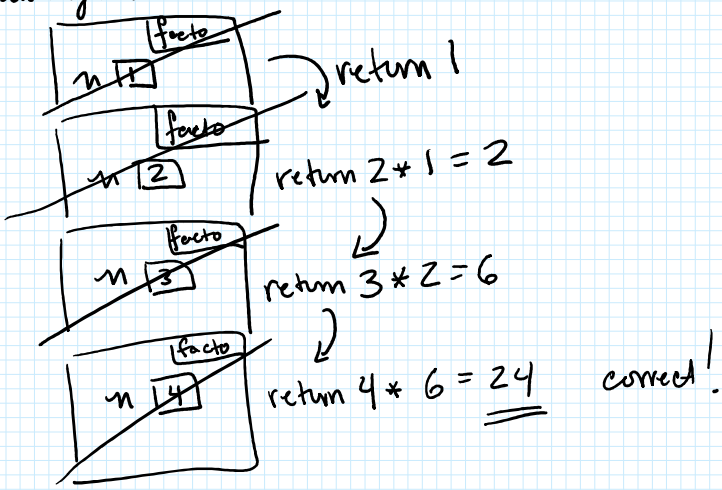
$0! = 1$ (by definition)

ex: $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$

We can write a recursive function to calculate factorial:

```
facto(n) : // assuming n is non-negative
    if n ≤ 1:
        return 1
    else:
        return n * facto(n-1)
```

If we call facto(4), this happens:



Let's prove that facto is correct by induction:

Begin by stating a recursive invariant:

P(n) = the function facto(n) returns n!.

Note: You must use the recursive variable n within your invariant.

① BASE CASE:

n = 0:

facto(0) returns 1

and 1 = 0! by definition ✓

n = 1:

pseudocode

```
facto(n) : // assuming n is non-negative
    if n ≤ 1:
    return 1
    else:
    return n * facto(n-1)
```

return n * facto(n-1)

n = 1:

facto(1) returns 1

and 1 = 1! by definition ✓

② INDUCTIVE HYPOTHESIS:

Assume for some integer k, P(k) is true: the function facto(k) returns k!.

③ INDUCTIVE STEP:

Want to show P(k+1): the function facto(k+1) returns (k+1)!.

facto(k+1) returns (k+1) · facto(k) by the pseudocode.

$(k+1) \cdot \text{facto}(k) = (k+1) \cdot k!$ by the inductive hypothesis

$= (k+1)!$ by math.

So facto(k+1) returns (k+1)! ✓

Another example algorithm

```
bool contains(array, n, value):
    if n < 0:
        return false
    else if array[n] == value:
        return true
    else:
        return contains(array, n-1, value)
```

Note: we discussed this in lab on Oct 6.

ex:

contains([6, 10, 11, 3], 3, 6) true

contains([6, 10, 11, 3], 3, 5) false

recursive calls have these parameters

array, 2, 6
array, 1, 6
array, 0, 6 ← returns true

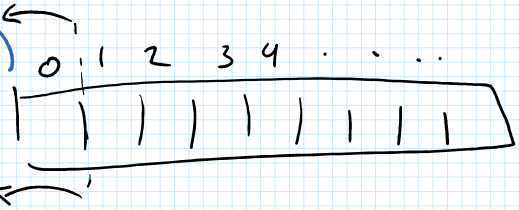
array, 2, 5
array, 1, 5
array, 0, 5
array, -1, 5 ← return false

Use induction to prove this function works properly:

First, state your RECURSIVE INVARIANT:

P(n) = "If the value occurs in the array at or before index n, the function returns true. Otherwise it returns false."

① BASE CASE: (note: we'll need several cases here)



n = 0:

If the value occurs in array at or before index 0, it must be at index 0.

According to the pseudocode, the "contains" fn returns true.

If the value does not occur at or before index 0, then it does not occur at index 0, so we go to the recursive case

an call contains(array, −1, value), which returns false, so we also return false.

② INDUCTIVE HYPOTHESIS

Assume $P(k)$ is true: "If the value occurs in the array at or before index k, the function returns true. Otherwise it returns false."

③ INDUCTIVE STEP

Want to show $P(k+1)$: "If the value occurs in the array at or before index k+1, the function returns true. Otherwise it returns false."

If the value is in the array at or before index k+1, then either:

① it's at index k+1. We return true immediately. ✓

OR

② it's at some earlier index. We call contains(array, k, value) and we return the same thing. By our inductive hypothesis, the recursive call is correct, and returns true. So we return true ✓

If the value is not in the array, so it's not at index k+1, so we recursively call contains(array, k, value). By our inductive hypothesis, this returns false so we return false. ✓